SLAC–PUB–4561 Rev.
March 1991
M

*INTRODUCTION TO HUMAN FACTORS**

Joan M. Winters
Stanford Linear Accelerator Center
Stanford University, Stanford, CA 94309

Introduction

There is much to be done in Human Factors. First, I'm going to give a brief overview of the field. I'll give some background on human factors, talk about what the nature of the problem is, what the costs are, how they can be alleviated, and why we don't just run out and fix them. Second, Paul Hoffman will discuss highlights from current research and some elements of the business case. Finally, Tamara Sturak, Manager of the Human Factors Project, will talk about activities of the Project.

Much of this material is adapted from the Report of the Interactive Systems Task Force (otherwise known as INTERSYS), on which I labored. The results were published as a SHARE strategic direction statement in *SHARE Secretary's Distribution* **#323**, October, 1982. IBM responded in March, 1984, after gathering views from people in a variety of areas

Presented at Session G200 Human Factors Project SHARE 71,
New York, New York, August 14-19, 1988

---

*Work supported by the Department of Energy, contract DE-AC03-76SF00515

across the corporation. Since INTERSYS attacked a broad set of subjects, the breadth of IBM's response was very appropriate; and we were happy to see IBM go to the effort. I have also drawn on information developed by the person who preceded me as Manager of the Human Factors Project, Jim Lipkis, and the IBM representative who encouraged the Project to create this talk, Dick Granda, as well as on many years of work with Project members, different IBM representatives to the Project, and others.

I should make a couple of *caveats:* I am not an expert in the field of human factors. We have at least one in the room, Ron Shapiro, our current IBM representative. These are my views after twelve years of being involved with the Human Factors Project and more than six working on a Usability Committee at Stanford Linear Accelerator Center. There we comment on commands that are being added to the VM system or enhanced.

The Field of Human Factors

Thomas Carlyle said, "Man is a tool-using animal.... Without tools he is nothing, with tools he is all." Human factors is concerned with designing more comfortable tools for humans to use. A definition of human factors is a field of applied psychology that studies attributes of the work or play environment with the goal of making them more conducive to human effectiveness. Human factors is concerned with relationships between human characteristics and behaviour and the "design of things people use, the methods of their use, and the environment in which they are used," as Michael Marcotty said at a SHARE General Session in 1981. The field is also known as ergonomics, particularly in Europe.

Human factors grew out of World War II attempts to design better airplane cockpits. The pilots kept pushing the wrong buttons, and the planes crashed. The field emerged in response. The initial practitioners tried to reduce failure rates, e.g., by designing cockpits so that the abort button wasn't right next to the start button. People studying hardware human factors have also investigated subjects like traffic signals and nuclear reactor control room design. In computing, the initial emphasis was likewise on hardware, particularly on the design of terminals and workstations and of furniture like chairs for using them. More recently has emerged the study of the intellectual and psychological factors of using software, or software human factors. For this, human factors draw on cognitive psychology. This branch of psychology attempts to understand humans' higher mental functions, such as learning, memory, reasoning, problem solving, and the usage of tools. While our understanding of physical human factors is reasonably good, we know few behavioural principles to apply to software human factors. This circumstance means that at this time the study of software human factors is an art rather than a science. The field is presently based on experimental data rather than predictive theories.

The goal of human factors is to contribute to the overall usability of systems, but usability and human factors are slightly different. Human factors are concerned with the design of the interface between the user and his or her data and between the user and his or her set of tools; whereas, usability is more general. It also addresses issues like function and system availability. Now there is a very gray area between function and human factors, for example, the Macintosh interface that allows multiple screens, bit-mapped displays, and the easy combination of graphics and text. This interface shows definite improvements in human factors but also represents significant increase in function.

The Costs of Poor Human Factors

So what's the problem? People use computers successfully. Why don't we just go on the way we have been?

We could. Humans are very adaptable. They will conform to just about anything, if they have to. But at a cost! Some of the problems with current systems I'll mention now. See Appendix A of the INTERSYS Report for an even more extensive, though still not comprehensive, list.

Today's systems are very large and complex, and getting more so. They lack conceptual integrity. Like Topsey, they've just grown. To their users, most systems seem arcane, idiosyncratic, inconsistent, and more difficult than necessary to learn and to use. This is a particular problem for occasional and discretionary users. As John Ehrman pointed out many years ago (in 1979), users need to learn many different interfaces and languages just to develop one application. In his paper on the Tower of Babel, he mentioned editors, programming languages (like FORTRAN and PL/I), command languages (like JCL, CMS, and CP), linking languages, debugging languages, document formatting languages (like SCRIPT), command procedure languages (at least we've got REXX now), other languages required by specific applications, and, of course, the natural language or languages of one's choice. (With VM/SP 5 IBM has even provided support for various natural languages, which is a good but difficult step.)

Another problem with interfaces is that they're often rigid and unnatural. This causes users to make errors at high rates. There are redundant names for the same function, leading people to try to uncover subtle differences when there are none. The error messages are often cryptic, assuming users have knowledge of system implementation. The user assistance facilities are often missing entirely or are available inconsistently and erratically. For example in VM, see DIRMAINT ? versus HELP ACCESS. Or with VS FORTRAN Interactive Debug, look at the radically different formats of those task files and the standard VM/SP task menus. Hardcopy documentation is often obsolete, incomplete, poorly written, and verbose. For many, it is rarely available when they want to get at it.

The costs of poor human factors in current systems can be divided into direct and indirect ones. These costs are impossible to quantify fully. Often they are distributed among diverse users off in isolated locations; whereas, the costs of buying a system or developing software up front are much more centralized. Due to their invisibility, user costs are underestimated.

Since the early fifties the tradeoff between machine and people costs has shifted from minimizing the costs of machines to minimizing the costs of people. Doherty and other researchers have estimated that the cost of user time wasted merely because of poor system response time is much greater than the cost of all the computer hardware, software, and support staff one might have.

Adapting categories of costs identified by Ledgard and Singer and also by Marcotty, I use three: direct, indirect, and opportunity costs.

In direct costs there are the cost of training and re-training, the time spent diagnosing and fixing errors, and the unnecessary reruns caused by error-prone aspects of the system and unhelpful error messages. In 1977, more than a decade ago, Gilb and Weinberg in *Humanized Input* estimated that at that point the unnatural choice of a blank delimiter in JCL had led by conservative estimate to the waste of $100,000,000 in unnecessary reruns, destroyed files, inefficient operation, and bug hunting. They also estimated that another $50,000,000 had been lost due to the positioning comma. De Greene cites another example of a NASA rocket that went off course and had to be destroyed to the tune of $35,000,000 because of a hyphen missing from its guidance equations.

Indirect costs include cognitive costs and human suffering. Some cognitive costs are the mental overload of remembering the many different languages and their idiosyncrasies, for example how to quit. Is it QUIT, EXIT, END, QQUIT, FINISH, *FIN*, BYE, or .... I'm

sure you know others. The bewildering array of command names for similar functions is made even more complex by the many different kinds of abbreviation rules.

Another type of cost is human suffering, elements of which include frustration, anxiety, pressure, fatigue, and fear. I'll give one example, of frustration. A number of years ago at a university that shall remain nameless, a professor who shall be unidentified was seen one Saturday morning in a public area jumping on a keypunch. Anxiety can come when the system does things you don't expect. Pressure and fatigue can be caused by frequent difficulties understanding obscure system error messages and by keeping track of the mentally-difficult exercises that systems demand. Then of course there's fear. Some studies have shown that when one talks of system commands new users think that the system is commanding them rather than that they should command the system.

Finally, there are opportunity costs, limits placed on future usage. Until a few years ago, most computer users were computer professionals. They had to use the computer, and they generally got into its obscurities and even liked them, feeling, for example, that they knew things other people didn't know. But now managers, professionals in many businesses, hobbyists, and home users — lots of people — are exploring computers. These discretionary users will only start and continue to use computers if the systems are easy enough to operate that these users find benefits quickly and permanently, enhancements to their ways of work and play that save them time, stimulate their imaginations, increase their overall effectiveness, and promote their satisfaction. Professionals are an especially-demanding audience. For example, in the early days of expert systems one was developed that was very good at diagnosing illnesses in patients. Unfortunately, doctors didn't like to use it. The reason turned out to be that doctors are trained in judgment and they didn't trust the answers they were being given by the expert system. That was the point at which the developers of expert systems realized they had to incorporate ways for the systems to explain their reasoning for particular conclusions to the human users.

Ideal System Interfaces

O.K., we have identified a problem. How can we alleviate the costs of poor human factors in computer systems?

At the highest level, we want systems that are graceful, ones that do not interfere with human thought processes. Even more, we'd like systems that augment human thought processes. System interfaces should be oriented toward users' tasks and terminologies. Systems should not force users into the perspectives of computers, except, of course, where appropriate, as for certain tasks belonging to system programmers.

This is a very important point: one can hardly ever say that anything is absolutely true in the area of software human factors. Context is very important.

Various ideal system characteristics that may aid in achieving the goal were identified by the INTERSYS group. These are not absolute, but characteristics that may be optimized and weighed against other considerations that may be present in a particular application.

Ideal Static Attributes:

The report identified eight static attributes of graceful system interfaces: simplicity, consistency, ease of recognition, ease of learning, integration, tailorability, robustness, and power.

Simplicity is the first static attribute. Simplicity has also been called parsimony. It means that designers need to minimize the number of operations that users must perform, the number of system-related concepts they need to learn, and the complexity and extraneous verbosity of system output. VM CMS HELP is mostly a counterexample. (I think,

supported by preliminary experience at SLAC, that the new VM/SP 5 BRIEF section contains insufficiently-specific information to be useful very often.) If the system knows the information, it shouldn't require users to re-enter it on input. If the system knows relevant information, it should tell users on output. It should not say, "Insufficient memory;" it should tell people how much more it needs. Better yet, the system should just go get the storage and not bother the users at all. There are trade-offs among simplicity, ease of recognition, ease of learning, and power.

The second static attribute is consistency. That means first that terminology should be consistent. The same syntax should be used whenever a particular semantic entity appears. The same semantic entity should be identified by the same syntax. As Teitelbaum (a previous IBM representative to the Project) showed, when the same information appears on display screens in a similar context, it should appear in the same place so that people can take advantage of prior learning to find it.

The third attribute is ease of recognition. This is sometimes called transparency. As reasonably possible, system designers should take advantage of users' pre-existing knowledge. The operation of the interface also should help users build a conceptual model of the system that is sufficiently coherent that they can extrapolate from one task how to do similar ones. For example, if one knows how to compile, load, and execute FORTRAN, it would be nice if one could make a successful guess at how to do the same with PASCAL (allowing for differences in names of the compilers and libraries). Once people have invested in learning something, they should be able to transfer that knowledge into analogous circumstances. This attribute is fostered by selection of recognizable dialogue keywords, probably a natural language subset, and consistent application of those keywords to specific functions. The ability to pronounce keywords aids in their recall. We also need the ability to tailor the vocabulary to users' pre-existing knowledge and natural languages.

The fourth static attribute is ease of learning. Interfaces can have all these desirable attributes, but they still need special aids to help users get started and fill in gaps in their understanding. Assistance facilities should take into consideration the three or so cognitive states that a survey done by the Online Documentation Committee of the SHARE Human Factors Project identified: learning, problem-solving, and refresh.

This committee was formed in August, 1979, at SHARE 53 in New York City. The group ultimately developed a questionnaire asking people to describe how each one solved a specific problem with using an interactive system. The questionnaire listed thirteen possible sources of information like online detail about an error message, online help for a command, online help about a subject area, hardcopy documentation, another person, and "Other." The questionnaire was distributed in 1981 by coordinators at twenty-eight institutions. 270 volunteers responded, anonymously. Of the questionnaires returned, we were able to use 229. Our IBM representatives at that time, Rich Halstead-Nussloch and Dick Granda, performed the first in-depth analysis, discovering that the descriptions of the particular problems could be classified into different cognitive states. These are learner, problem-solver, and refresher. The states had been identified by research in cognitive psychology.

The learner has to learn at least some new concept, relationship, and/or nomenclature in order to work through a situation. A learner may be someone trying to use computing for the first time or someone starting in a new area, like a knowledgeable MVS/TSO user moving to VM/CMS.

A problem solver knows all the critical concepts, relationships, and nomenclature but does not know how they fall together to resolve the specific problem. A problem solver could be a statistician figuring out how to save recoded variables in SPSS or someone debugging a program.

A refresher has resolved a similar situation in the past but needs reminding of particular aspects or details. A refresher may reread documentation on a mail system to recall relationships among its components or look at the HELP for COPYFILE to find an infrequently-used option like TRANS.

As the survey showed, people in these separate states use different sources of information and are satisfied differently by those sources. For example, refreshers used online help for commands less frequently than learners but were more satisfied. In fact, neither learners nor problem solvers found much satisfaction in most nonhuman information sources. Until designers can incorporate more aspects of person-to-person communication into nonhuman information sources, they will not be very effective in helping people in the learning and problem solving states to satisfy their questions.

Expert systems offer some hope of meeting the need. Also, systems should keep enough information on users, their recent actions, general levels of expertise, and preferences, so that the software aids learning by tailoring messages and other displays to people's current computing status, cognitive states, backgrounds, and styles. SPIRES HELP provides the first of these functions. SPIRES is a data base system, and its HELP command displays what file is selected, how many matches were identified by the last query, and other context-dependent information.

The fifth attribute is integration. The entire system should start out and remain coherent to users. Different parts of the system should work well together. Users should be able to combine parts easily, even in ways designers never anticipated. As the system develops over time, it should not become "fragmented." This was a term identified by the SHARE LSRAD Task Force, which studied large systems requirements for application development. (INTERSYS was a child of LSRAD.) As each system evolves, people should be able to add new tools that integrate smoothly into what's already there.

The sixth attribute is tailorability. People have different styles, as well as general and specific knowledge, computing expertise, and computing tasks. Users need session and application defaulting, aliasing, and overriding so that they can develop environments comfortable to themselves and maintain consistency within their conceptual models. Users also need easy-to-learn and powerful EXEC languages like REXX so that they can combine several commands into one (incorporating some logic) to reduce their memory load. Once their environments are tailored to their satisfaction, users may need to "port" their views to multiple systems (like mainframe and pc). IBM's System Applications Architecture (SAA) may increase consistency across operating systems, but will it support enough tailorability and ways of easily porting one's tailorings that we can say SAA has this attribute? Perhaps we'll learn at one of the many SAA sessions this week.

The seventh attribute is robustness. Systems shouldn't break unpleasantly. They shouldn't crash or hang; and, most particularly, they shouldn't destroy data. Systems should be somewhat tolerant of minor differences in the way requests are specified. For example, CMS is robust in that it recognizes the minimum abbreviations of commands or any longer versions of the words, so that users don't have to remember specific abbreviations. PLATO (an educational system developed at the University of Illinois) is even better. It recognizes certain spelling errors and corrects for them, thus removing some of the irritation of "typo-ed" commands from users.

The final static attribute identified by INTERSYS is power. This provides for the easy combination of data and processing power from a wide variety of sources, including from machines other than the ones to which the users are primarily connected, even from ones produced by vendors other than IBM that obey different protocols. IBM's Extended Connectivity Facility (ECF), Advanced Program to Program Communications (APPC),

and related products may help enable such combinations. Power also requires support for concurrent asynchronous functions, so that one can lay something aside, handle an interrupt or pursue a thought or three, and return to what one was doing without losing any work. Certainly XEDIT's ability to pass commands through to VM and even be re-invoked itself is a step towards providing power. XEDIT now creates a much more effective work environment than when it was limited to CMS Subset. Windows may be another aid in this area. Without these appropriately powerful capabilities, users are forced into time-wasting behaviours attempting to bypass the limitations.

Ideal Dynamic Attributes:

The INTERSYS Task Force also identified two dynamic attributes of graceful system interfaces, to which I've added three more.

The first dynamic attribute is response time. This has been defined as the elapsed time between a user request and the system response. Response time should be fast enough not to interrupt human thought processes. Research by Doherty and Thadhani of IBM indicates that sub-second response time is needed. Doherty now thinks that times, probably down to a tenth of a second, are required. Both have shown that productivity as measured by the number of transactions per unit time increases in more than direct proportion to decreases in response time, e.g., transaction productivity goes up very quickly when response time is faster than a second. Subsequent research by Lambert where IBM provided sub-second response time to a group of developers showed additional benefits.

IBM has a standard way of estimating development projects, e.g, how long a project should take, how many programmers it needs, and how many errors the finished code is likely to have. Comparing the predicted values with the actual ones, Lambert concluded that for their response time of 0.84 seconds (reduced from 2.3 seconds), the duration of the project decreased by 21%, the amount of programmer time was 39% less that predicted, productivity (considering both the size and complexity of the program) increased 58%, and quality (in terms of errors reported) remained at a small number (eight). In addition, the program developers were happier than before.

The reason seems to be related to people's attention spans. People usually don't think after each command. They "chunk" commands. Humans seem to have sequences of actions in mind, contained in their short term memory buffers; and they just want to enter the commands one right after the other before they stop to analyze the output. If something interrupts them, it's likely to blow away their short term memory buffers so that they have to recreate their action lists. Restarting takes time and is irritating to people.

At IBM's T. J. Watson Research Center, there's been a goal of providing sub-second response time for a number of years now. Even several years ago, they achieved this aim for $8000 per year per professional working on the computing system, while the user salary costs, only while the workers were sitting at their workstations, was $41,000 per year. Doherty and Pope, looking at the environment of 1984, concluded that "the value [derived from corporate gross annual income] of the active human time at the keyboard for the human-intensive work was 35 times greater than the cost of the computing service."

The second dynamic attribute identified by INTERSYS is system lockout. Some people have a model of human/computer interaction as being a polite dialogue. The human says something, the keyboard locks up, the computer processes, the computer displays its response, the keyboard unlocks, and the human can say something else. Again, this sequence seems to interrupt humans' short term memory buffers. Often people in a conversation like to interrupt each other or talk altogether at once, so user lockout interferes with graceful human/computer interfaces.

The third dynamic attribute of a graceful interface is bandwidth. Both INTERSYS and Doherty have emphasized its importance. Sub-second response time is not enough. People need to transmit ever more data between their workstations and other parts their systems. For example, one may look at a graphics display, make a few small changes in light of what one sees, and then request an updated version in pursuit of the effects of the changes. The successive iterations should be displayable very quickly, so that people's short term memory buffers, which are guiding the explorations, don't get blown away. To do this kind of display and other sorts of applications like scanning requires much broader bandwidths than are common today.

The fourth dynamic attribute is feedback. Interfaces should keep users informed about the status of whatever's happening. As I mentioned, SPIRES HELP provides feedback. So do a number of VM/SP QUERY commands and other software you can think of.

The final dynamic attribute is locus of control. This treats whether the people's modes of interaction with computer systems are active or passive. Some people like to feel in control of their computing sessions; others prefer to be led along. I think this difference relates to their cognitive states. Learners are more likely to be in a passive state; refreshers in an active state.

Difficulties in Fixing System Interfaces

So what's the problem? Why don't we just go out, incorporate knowledge from the field of human factors, and make our system interfaces graceful?

It's not that clear how to do it. As Thomas More said, "If you had been with me in Utopia ... you would frankly confess that you had never seen a people ordered so well as they were." That's fine in Utopia, but we don't have Utopia here. The characteristics I've talked about are just that, Utopian. We're not even sure these attributes will help, though I'm fairly sure they would; but it's not clear how to trade them off, and there may be other factors we haven't identified yet.

It turns out that intuition is of little help in creating system interfaces with good human factors. Moran pointed out that once designers become interested in human factors, the designers usually feel that they are human and thus can predict what's easy for people to use. The problem is that designers' intuitions don't necessarily (or even usually) match users'. They are from different audiences with different backgrounds, conceptual models, and tasks and are often in different cognitive states.

Also, when one is implementing actual systems, one has to face difficult tradeoffs because of limits in the amount of time, money, and other resources one can spend on the efforts. There are not yet any algorithms to predict how to make the tradeoffs for any given set of circumstances.

For example, we want consistency on the one hand and tailorability on the other. Standards activities push various forms of consistency, but people feel comfortable and are presumably more productive if they can make their systems feel right to them. Which is more important depends on the situation, the audiences, etc. There is no general answer, though the answer may often be both, if you can afford it.

We want systems to be powerful, robust, and integrated, even after being extended; and then, of course, we want them to be simple.

Part of the answer depends on identifying who the users are. As I've mentioned, they have different tasks, professional knowledge, knowledge of computing, personal styles, intelligence, cognitive states, and organizational requirements. The needs of end users are often different from those of computer center staff. It's hardest to design a general interface for a multiplicity of audiences.

Research has shown that the numerous features of even simple experiments interact in complex ways on subjects, so that it is very difficult to build general models of user behaviour. It's easy to over-generalize or misinterpret the results or to create an experimental design that is inadequate.

A number of years ago, there was an experiment that tried to determine whether having comments in code was desirable or not. The experimenters thought comments would be helpful, but the initial analysis indicated that they weren't. Looking into the experiment further showed that adding the comments had increased the size of the test program over a page boundary, so that the subjects had to keep flipping between the pages and that interruption of their thoughts was interfering with their overall comprehension of the program. What this experiment finally showed was the importance of context, of presenting information in ways where people can see the overall structure, rather than the "badness" of comments.

In Conclusion

I've given some background on the field of human factors, talked about the nature of problems with current human/computer interfaces, identified some of their costs, outlined ideal attributes of graceful system interfaces, and indicated some reasons why it's not easy to fix the problems. Now I'll turn the floor over to Paul Hoffman, who will give us reason to hope that we can improve these interfaces.

This work was supported by the Department of Energy under contract number DE-AC03-76SF00515.

Bibliography

Chapman, J. A., *et al.,* 1982. "Computing for the Information Age: The Report of the **Interactive Systems** Task Force (INTERSYS)," *SHARE Secretary's Distribution* **323.**

Cowlishaw, M. F., 1983. "Design of the Restructured Extended Executor Language," TR 12.223 (IBM United Kingdom Laboratories Limited, Hursley Park, Winchester, Hampshire, England SO212JN).

De Greene, Kenyon B., 1970. "Systems and Psychology," *Systems Psychology,* Kenyon B. De Greene, editor (New York).

Doherty, W. J., and R. P. Kelisky, 1979. "Managing VM/CMS systems for user effectiveness," *IBM Systems Journal* **18**(1), 143–163.

Doherty, W. J., and W. G. Pope, 1986. "Computing as a tool for human augmentation," *IBM Systems Journal* **25**(3/4), 306–320.

Ehrman, John R., 1979. "The Babel of Application Development Tools," *Proceedings of SHARE 53* **I** (SHARE Inc., Chicago), 29–38.

Engel, Stephen E., and Richard E. Granda, 1975. "Guidelines for Man/Display Interfaces," TR 00.2720 (IBM Poughkeepsie Laboratory, Poughkeepsie, N.Y.)

Gilb, Tom, and Gerald M. Weinberg, 1977. *Humanized Input: Techniques for Reliable Keyed Input* (Cambridge, Mass.)

Guynes, Jan L., 1988. "Impact of System Response Time on State Anxiety," *Communications of the ACM* **31**(3), 342–347.

IBM Corporation, 1982. "The Economic Value of Rapid Response Time," IBM Form Number GE20-0752.

IBM Corporation, 1987. *Systems Application Architecture: Common User Access Panel Design and User Interaction*, IBM Form Number SC26-4351.

Lambert, G. N., 1984. "A comparative study of system response time on program developer productivity," *IBM Systems Journal* **23**(1), 36–43.

Ledgard, Henry, and Andrew Singer, 1978. "The Case for Human Engineering," *Proceedings of SHARE 50* **III** (SHARE Inc., Chicago), 1447–1466.

LSRAD Task Force, 1979. *Towards More Usable Systems: The LSRAD Report: Large Systems Requirements for Application Development* (SHARE Inc., Chicago)

Marcotty, Michael, 1981. "Human Factors and Productivity," *Proceedings of SHARE 56* **I** (SHARE Inc., Chicago), 328–337.

Markell, R. A., and L. H. Fenton, 1984. "Response to the SHARE Interactive Systems Task Force Report (INTERSYS)," *Proceedings of SHARE 62* **I** (SHARE Inc., Chicago), 865–879.

Miller, George A., 1956. "Information and Memory," *Scientific American* (August).

Miller, James Grier, 1978. *Living Systems* (New York).

Moran, Thomas P., 1981. "Guest Editor's Introduction: An Applied Psychology of the User," *ACM Computing Surveys* **13**(1), 1–11.

Smith, Sidney L., and Jane N. Mosier, 1984. *Design Guidelines for User-System Interface Software,* ESD-TR-84-190 MTR-9420 (The Mitre Corporation, Bedford, Mass.)

Teitelbaum, Richard C., *et al.*, 1983. "The Effects of Positional Constancy on Searching Menus for Information," TR 00.3248 (IBM Poughkeepsie Laboratory, Poughkeepsie, N. Y.)

Thadhani, A. J., 1981. "Interactive user productivity," *IBM Systems Journal* **20**(4), 407–423.

Winters, Joan M., 1978. "Human Factors Considerations in the Design of Vendor-supplied Documentation," *Proceedings of SHARE 51* **I** (SHARE Inc., Chicago), 536–541.

——, 1983. "The User and Effectiveness of Online Documentation Facilities — An Empirical Study," *Proceedings of SHARE 60* **II** (SHARE Inc., Chicago), 798–852.

——, and Keith J. Sours, 1983. "An Empirical Study of the Use and Effectiveness of Online Documentation: Final Report," *Proceedings of SHARE 61* **I** (SHARE Inc., Chicago), 13–46.